3. Date abstracte în Modula 2.

- La baza descrierii datelor abstracte stă structura de **modul**.

- Modul este compuse din :
  - **modulul de definiție** (definition module) – conține declarațiile entităților exportate de către modul. Tot ce este declarat în modulul de definitie este vizibil în exterior.
  - **modulul de implementare** (implementation module) – conține detaliile concrete de implementare a obiectelor declarate în modulul de definiție. Nimic din ceea ce conține modulul de implemetare nu este vizibil în exterior.

- Există posibilitatea compilării separate a modulelor , cu păstrarea riguroasă a verificării tuturor compatibilirăților de tip între obiectele programului, indiferent de modulul în care au fost declarate.

- Tipul precum şi titlurile subprogramelor reprezentând operații vor fi specificate în modulul de definiție, iar modulul de implementare va conține declararea completă a subprogramelor care implementează operațiile.

- Descrierea unui tip abstract de stivă în Modula 2:

```
DEFINITION MODULE stack;
    CONST nr_max=100;
    TYPE stiva= RECORD
                    tab_st:ARRAY[1..nr_max] OF INTEGER;
                    ind:INTEGER;
                END;
    PROCEDURE  pop( VAR s:stiva ) : INTEGER;
    PROCEDURE  push( VAR s:stiva , x : INTEGER );
    PROCEDURE  top(  s:stiva ) : INTEGER;
    PROCEDURE  init ( VAR s:stiva );
END stack;


IMPLEMENTATION MODULE stack;
    PROCEDURE  empty(  s:stiva ) : BOOLEAN;
    BEGIN
        RETURN s.ind=1;
    END empty;
    PROCEDURE  overflow(  s:stiva ) : BOOLEAN;
    BEGIN
        RETURN s.ind > nr_max;
    END overflow;
    PROCEDURE  pop( VAR s:stiva ) : INTEGER;
    BEGIN
        IF NOT empty(s) THEN
            s.ind:=s.ind-1;
            RETURN s.tab_st[s.ind];
        ELSE
            HALT;
        END;
    END pop;
```

```
    PROCEDURE  push( VAR s:stiva , x:INTEGER );
    BEGIN
        IF NOT overflow(s) THEN
            s.tab_st[s.ind]:=x;
            s.ind:=s.ind+1;
        ELSE
            HALT;
        END;
    END push;

    PROCEDURE  top(  s:stiva ) : INTEGER;
    BEGIN
        IF NOT empty(s) THEN
            RETURN s.tab_st[s.ind-1];
        ELSE
            HALT;
        END;
    END top;

    PROCEDURE  init( VAR s:stiva ) ;
    BEGIN
        s.ind:=1;
    END init;

BEGIN
END stack.
```

- Descrierea unui modul ce operează cu o stivă :

```
MODULE st;
    FROM stack IMPORT stiva, init, push, pop, top;
    FROM IO IMPORT WrStr, WrLn, WrInt, RdInt, RdStr;
    VAR x:INTEGER;
        s:ARRAY[1..50] OF CHAR;
        s1,s2:stiva;
BEGIN
    init(s1); init(s2);
    REPEAT
        WrLn;  WrStr("comanda:"); RdStr(s);
        CASE s[1] OF
            'p':WrStr("valoarea:");
                x:=RdInt();
                IF s[2]='1' THEN
                        push(s1,x);
                ELSE
                        push(s2,x);
                END |
            't': IF s[2]='1' THEN
                        WrInt(top(s1),4);
                ELSE
                        WrInt(top(s2),4);
                END |
            'r': IF s[2]='1' THEN
                        WrInt(pop(s1),4);
                ELSE
                        WrInt(pop(s2),4);
                END |
            'f': |
            ELSE WrStr("comanda eronata")
        END;
    UNTIL s[1]='f';
END st.
```

- Observații:
  - Modula 2 nu permite parametrizarea tipului abstract.
  - Dacă un tip este declarat într-un modul de definiție, toate detaliile privind structura acestuia sunt vizibile din modulele importatoare ale tipului (este exportat în mod "transparent"). Se acceptă instrucțiunile:
    s1.ind:=1000; s2.tab_st[3]:=0;
- În Modula 2 există posibilitatea de a exporta tipuri în mod "**opac**". În modulul de definiție se declară doar numele tipului; descrierea tipului se "ascunde" în modulul de implementare. Exemplu:

```
DEFINITION MODULE stack;
    TYPE stiva;
    PROCEDURE  pop( VAR s:stiva ) : INTEGER;
    PROCEDURE  push( VAR s:stiva , x : INTEGER );
    PROCEDURE  top(  s:stiva ) : INTEGER;
    PROCEDURE  init ( VAR s:stiva );
END stack.
IMPLEMENTATION MODULE stack;
    FROM Storage IMPORT ALLOCATE;
    CONST nr_max= 100;
    TYPE stv=RECORD
                    tab_st:ARRAY[1..nr_max] OF INTEGER;
                    ind:INTEGER; END;
        stiva=POINTER TO stv;
    PROCEDURE  empty( VAR s:stiva ) : BOOLEAN;
    BEGIN
        RETURN s↑.ind=1;
    END empty;
    PROCEDURE  overflow( VAR s:stiva ) : BOOLEAN;
    BEGIN
        RETURN s↑.ind > nr_max;
    END overflow;
```

```
    PROCEDURE  pop( VAR s:stiva ) : INTEGER;
    BEGIN
        IF NOT empty(s) THEN
            s↑.ind:=s↑.ind-1;
            RETURN s↑.tab_st[s↑.ind];
        ELSE
            HALT;
        END;
    END pop;
    PROCEDURE  push( VAR s:stiva , x:INTEGER );
    BEGIN
        IF NOT overflow(s) THEN
            s↑.tab_st[s↑.ind]:=x;
            s↑.ind:=s↑.ind+1;
        ELSE
            HALT;
        END;
    END push;
    PROCEDURE  top(  s:stiva ) : INTEGER;
    BEGIN
        IF NOT empty(s) THEN
            RETURN s↑.tab_st[s↑.ind-1];
        ELSE
            HALT;
        END;
    END top;
    PROCEDURE  init( VAR s:stiva ) ;
    BEGIN
        ALLOCATE(s,SIZE(stv));
        s↑.ind:=1;
    END init;
BEGIN
END stack.
```

4. Date abstracte în Ada.

- Datele abstracte sunt descrise cu ajutorul structurii de pachet (**package**).
- Pachetul Ada este alcătuit din:
- **Specificarea pachetului** – conține:
- declararea entităților vizibile din exteriorul pachetului;
- declarare de entități inaccesibile din exterior (porțiune privată).
- **Corpul pachetului** – conține detaliile concrete de implementare, invizibile din exterior.

- **Descriere tipului abstract *stiva.***

```
package stack is
    type stiva(nr_max:integer) is
        record
            tab_st:array(1..nr_max) of integer;
            ind:integer;
        end record;
    function pop(s: in out stiva) return integer;
    procedure push(s: in out stiva; x: integer);
    function top(s: stiva) return integer;
    procedure init(s: out stiva);
end stack;
package body stack is
    function  empty(s:stiva ) return boolean is
    begin
        return  s.ind=1;
    end empty;
    function  overflow(s: stiva ) return boolean is
    begin
        return  s.ind>s.nr_max;
    end overflow;
```

```
function pop(s:in out stiva ) return integer is
begin
   if not empty(s) then
      s.ind:=s.ind-1;
      return s.tab_st(s.ind);
   else
      raise eroare;
   end if;
end pop;
procedure push(s:in out stiva; x: integer) is
begin
   if not overflow(s) then
      s.tab_st(s.ind):=x;
      s.ind:=s.ind+1;
   else
      raise eroare;
   end if;
end push;
function top(s:stiva ) return integer is
begin
   if not empty(s) then
      return s.tab_st(s.ind-1);
   else
      raise eroare;
   end if;
end top;
procedure  init(s:out stiva ) is
begin
      s.ind:=1;
end init;

begin
end stack;
```

- Utilizarea pachetului *stack*:

```
procedure st is
    use stack;
    st1:stiva(100);   st2:stiva(50);
    ----------------------------
    init(st1);  init(st2);
    ----------------------------
    push(st1,10);
    ----------------------------
    push(st2,5);
    ----------------------------
    i:=top(st1);
end st;
```

În procedura *st* sunt permise operații ca:

```
st1.ind := 1000;
st2.tab_st(3) := 0;
```

Soluția: tipul *stiva* declarat ca privat.

```
package stack is
    type stiva(nr_max:integer) is private;
    function pop(s: in out stiva) return integer;
    procedure push(s: in out stiva; x: integer);
    function top(s: stiva) return integer;
    procedure init(s: out stiva);
private
     type stiva(nr_max:integer) is
          record
             tab_st:array(1..nr_max) of integer;
             ind:integer;
          end record
end stack;
```

- Tipul *stiva* poate fi declarat ca **privat limitat (limited private).** Asupra obiectelor de acest tip pot fi aplicate exclusiv operaţii exportate din pachetul care exportă şi tipul limitat.

- **Declararea pachetelor generice –** o facilitate foarte importantă a limbajului Ada:

```
generic
        nr_max:integer;
        type tip_el is private;
package stack is
        type stiva is limited private;
        function pop(s:in out stiva) return tip_el;
        procedure push(s:in out stiva; x:tip_el);
        function top(s:stiva) return tip_el;
        procedure init(s:in out stiva);
private
        type stiva is
                record
                   tab_st:array(1..nr_max) of tip_el;
                   ind:integer;
                end record;
end stack;
```

```
package body stack is
    function empty(s:stiva) return boolean is
    begin
        ----------
    end  empty;
    function overflow(s:stiva) return boolean is
    begin
        return s.ind > nr_max;
    end  overflow;
    function pop(s: in out stiva) return tip_el  is
    begin
        ----------
    end  pop;
    procedure push(s: in out stiva; x:tip_el) is
    begin
        if not overflow(s) then
            s.tab_st(s.ind):=x;
            s.ind:=s.ind+1;
        else
            raise eroare;
        end if;
    end  push;
    function top(s:stiva) return tip_el is
    begin
        ----------
    end  top;

    function init(s: out stiva)  is
    begin
        ----------
    end  init;
begin
end stack;
```

- Pachetul generic poate fi instanțiat pentru orice tip de element şi orice dimensiune de stivă:

```
procedure st is
    type t is …;
    package stack_t is new stack(100,t);
    use stack_t;
    stv:stiva;
    x:t;
    ----------------------
    init(stv);
    ----------------------
    push(stv,x);
    ----------------------
    x:=top(stv);
    ----------------------
end st;
```